Advanced Database in PHP

David T. Makota
Faculty of Computing, Information Systems & Mathematics
The Institute of Finance Management
Dar Es Salaam, Tanzania

Advanced Database in PHP

Contents

Contents

- Databases and Web Development
- Database APIs
- Transactions
- Accessing MySQL in PHP

Advanced Database in PHP

Lecture iii

Advanced Database in PHP

Databases and Web Development

Databases and Web Development

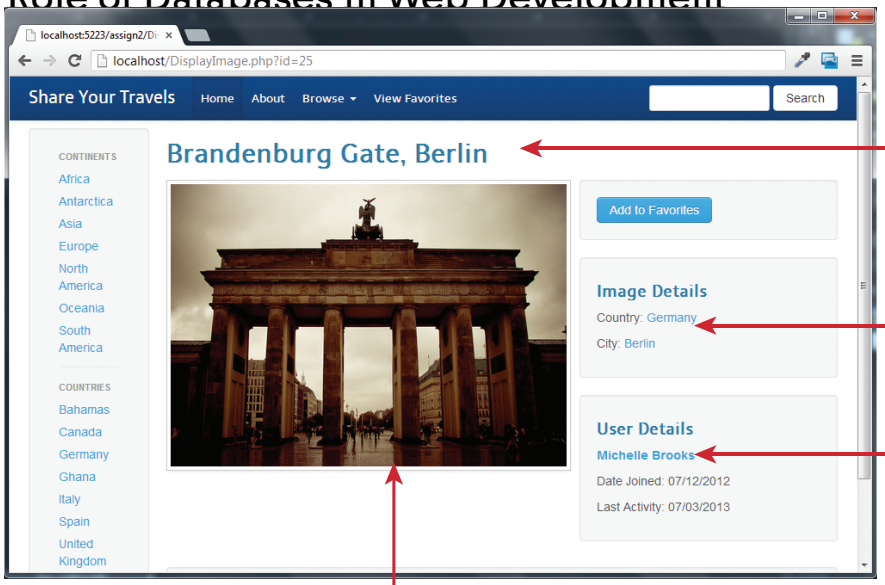
The Role of Databases in Web Development

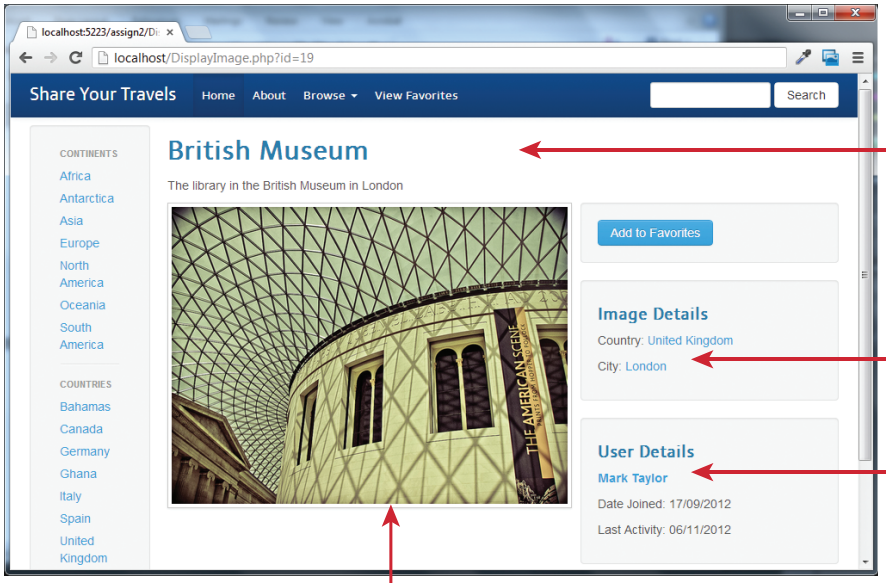
Databases provide a way to implement one of the most important software design principles namely, that:

> *one should separate that which varies from that which stays the same .*

Databases and Web Development

The Role of Databases in Web Development

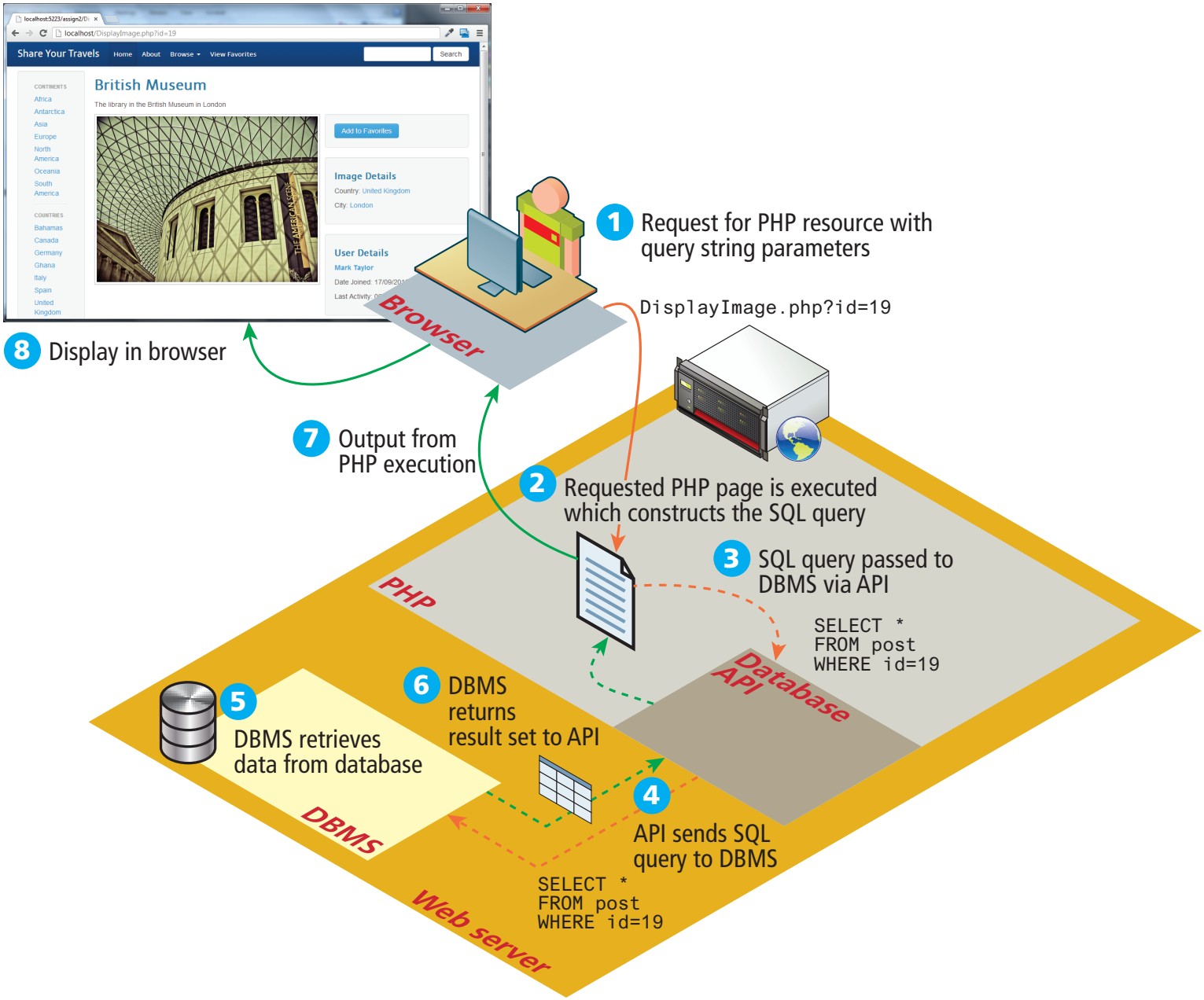




Content (data) varies but the markup (design) stays the same.

Databases and Web Development

How websites use databases



Advanced Database in PHP

Database APIs

Database APIs

PHP MySQL APIs

- **MySQL extension**. This was the original extension to PHP for working with MySQL and has been replaced with the newer mysqli extension.
- **mysqli extension**. This extension provides both a procedural and an object-oriented approach. This extension also supports most of the latest features of MySQL.
- **PHP data objects (PDOs)**. provides an abstraction layer that with the appropriate drivers can be used with any database, and not just MySQL databases. However, it is not able to make use of all the latest features of MySQL.

Database APIs

Deciding on a Database API

While PDO is unable to take advantage of some features of MySQL, there is a lot of merit to the fact that PDO can create database-independent PHP code

- Like many things in the web world, there is no single best choice.
- As we proceed, we will standardize on the object-oriented, database-independent PDO approach.

Advanced Database in PHP

Transactions

Transactions

An Advanced Topic.

Anytime one of your PHP pages makes changes to the database via an UPDATE, INSERT, or DELETE statement, you also need to be concerned with the possibility of failure.

A **transaction** refers to a sequence of steps that are treated as a single unit, and provide a way to gracefully handle errors and keep your data properly consistent when errors do occur.

Transactions

An Example

Imagine how a purchase would work in a web storefront. After the user has verified the shipping address, entered a credit card, and selected a shipping option and clicks the final *Pay for Order* button? Imagine that the following steps need to happen.

1. Write order records to the website database.
2. Check credit card service to see if payment is accepted.
3. If payment is accepted, send message to legacy ordering system.
4. Remove purchased item from warehouse inventory table and add it to the order shipped table.
5. Send message to shipping provider.

Transactions

An Example

At any step in this process, errors could occur. For instance:

- The DBMS system could crash after writing the first order record but before the second order record could be written.
- The credit card service could be unresponsive, or the credit card payment declined.
- The legacy ordering system or inventory system or shipping provider system could be down.

Transactions

Multiple types

Local Transactions can be handled by the DBMS.

Distributed Transactions involve multiple hosts, several of which we may have no control over.

Distributed transactions are much more complicated than local transactions

Local Transactions

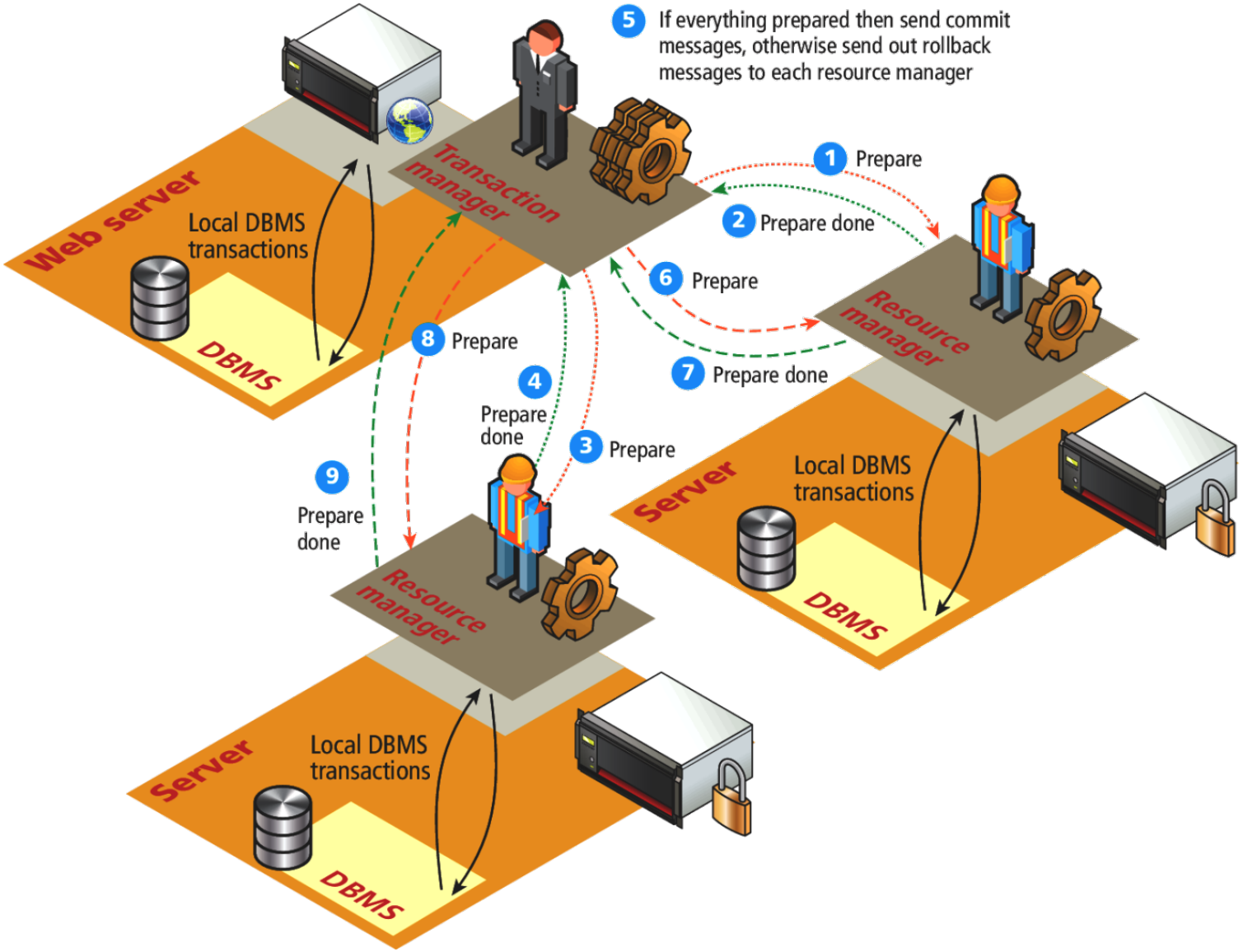
The easy transactions

The SQL for transactions use the START TRANSACTION, COMMIT, and ROLLBACK commands

```
/* By starting the transaction, all database modifications within
   the transaction will only be permanently saved in the database
   if they all work   */

START TRANSACTION

INSERT INTO orders ...
INSERT INTO orderDetails ...
UPDATE inventory ...

/* if we have made it here everything has worked so commit changes */
COMMIT

/* if we replace COMMIT with ROLLBACK then the three database
   changes would be "undone" */
```

Distributed Transactions

Depends on which external systems…



Advanced Database in PHP

Accessing MySQL in PHP

Accessing MySQL in PHP

Basic Connection Algorithm

1. Connect to the database.
2. Handle connection errors.
3. Execute the SQL query.
4. Process the results.
5. Free resources and close connection.

Accessing MySQL in PHP

Basic Connection Algorithm

```
<?php

try {
   $connString = "mysql:host=localhost;dbname=bookcrm";
   $user = "testuser";
   $pass = "mypassword";

   $pdo = new PDO($connString,$user,$pass);
   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   $sql = "SELECT * FROM Categories ORDER BY CategoryName";
   $result = $pdo->query($sql);

   while ($row = $result->fetch()) {
      echo $row['ID'] . " - " . $row['CategoryName'] . "<br/>";
   }
   $pdo = null;
}
catch (PDOException $e) {
   die( $e->getMessage() );
}

?>
```

1
2
3
4
5

Accessing MySQL in PHP

Connecting to a Database (mysqli procedural)

```
// modify these variables for your installation
$host = "localhost";
$database = "bookcrm";
$user = "testuser";
$pass = "mypassword";
$connection = mysqli_connect($host, $user, $pass, $database);
```

Accessing MySQL in PHP

Connecting to a Database (PDO Object-oriented)

```
// modify these variables for your installation
$connectionString =
"mysql:host=localhost;dbname=bookcrm";
$user = "testuser";
$pass = "mypassword";
$pdo = new PDO($connectionString, $user, $pass);
```

Accessing MySQL in PHP

Handling Connection Errors - mysqli

```
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

// mysqli_connect_errno returns the last error code

if ( mysqli_connect_errno() ) {

        die( mysqli_connect_error() );
        // die() is equivalent to exit()

}
```

Accessing MySQL in PHP

Handling Connection Errors - PDO

```
try {
        $connString = "mysql:host=localhost;dbname=bookcrm";
        $user = DBUSER;
        $pass = DBPASS;
        $pdo = new PDO($connString,$user,$pass);
        . . .
}
catch (PDOException $e) {
        die( $e->getMessage() );
}
```

Accessing MySQL in PHP

Executing the Query

```
$sql = "SELECT * FROM Categories ORDER BY CategoryName";

// returns a mysqli_result object

$result = mysqli_query($connection, $sql);

OR

$result = $pdo->query($sql);
```

Accessing MySQL in PHP

Processing the Query Results

```
$sql = "SELECT * FROM Categories ORDER BY CategoryName";

// run the query

$result = $pdo->query($sql);

// fetch a record from result set into an associative array

while ($row = $result->fetch()) {

        // the keys match the field names from the table

        echo $row['ID'] . " - " . $row['CategoryName'];

        echo "<br/>";

}
```

Accessing MySQL in PHP

Processing the Query Results

```
$sql = "select * from Paintings";
$result = $pdo->query($sql);
```

ID	Title	Artist	Year
345	The Death of Marat	David	1793
400	The School of Athens	Raphael	1510
408	Bacchus and Ariadne	Titian	1520
425	Girl with a Pearl Earring	Vermeer	1665
438	Starry Night	Van Gogh	1889

`$result`
Result set is a type of cursor to the retrieved data

```
$row = $result->fetch()
```

ID	Title	Artist	Year
345	Death of Marat	David	1793

keys

values

`$row`
Associative array

Accessing MySQL in PHP

Freeing Resources and Closing Connection

```
//closes the connection

mysqli_close($connection);

// closes connection and frees the resources used by the PDO object

$pdo = null;
```

Accessing MySQL in PHP

Working with Parameters – Technique 1 ? Placeholders

```
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear,
ImprintId,
ProductionStatusId, TrimSize, Description) VALUES (?,?,?,?,
?,?,?)";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $_POST['isbn']);
$statement->bindValue(2, $_POST['title']);
$statement->bindValue(3, $_POST['year']);
$statement->bindValue(4, $_POST['imprint']);
$statement->bindValue(5, $_POST['status']);
$statement->bindValue(6, $_POST['size']);
$statement->bindValue(7, $_POST['desc']);
$statement->execute();
```

Accessing MySQL in PHP

Working with Parameters – Technique 1 ? Placeholders *with Array*

```
/* can pass an array, to be used in order */

$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear,
ImprintId,
ProductionStatusId, TrimSize, Description) VALUES (?,?,?,?,
?,?,?)";
$statement = $pdo->prepare($sql);
$statement->execute array(array($_POST['isbn'],
$_POST['title'],$_POST['year'], $_POST['imprint'],
$_POST['status'], $_POST['size'],$_POST['desc']));
```

Accessing MySQL in PHP

Working with Parameters – Technique 2 - named parameters

```
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear,
ImprintId,
ProductionStatusId, TrimSize, Description) VALUES (:isbn,
:title, :year, :imprint, :status, :size, :desc) ";
$statement = $pdo->prepare($sql);
$statement->bindValue(':isbn', $_POST['isbn']);
$statement->bindValue(':title', $_POST['title']);
$statement->bindValue(':year', $_POST['year']);
$statement->bindValue(':imprint', $_POST['imprint']);
$statement->bindValue(':status', $_POST['status']);
$statement->bindValue(':size', $_POST['size']);
$statement->bindValue(':desc', $_POST['desc']);
$statement->execute();
```

Accessing MySQL in PHP

Working with Parameters – Technique 2 - named parameters *with Array*

```
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear,
ImprintId,
ProductionStatusId, TrimSize, Description) VALUES (:isbn,
:title, :year, :imprint, :status, :size, :desc) ";
$statement = $pdo->prepare($sql);
$statement->execute(array(':isbn' => $_POST['isbn'],
                          ':title'=> $_POST['title'],
                          ':year'=> $_POST['year'],
                          ':imprint'=>  $_POST['imprint'],
                          ':status'=> $_POST['status'],
                          ':size'=> $_POST['size']
                          ':desc'=> $_POST['desc']));
```

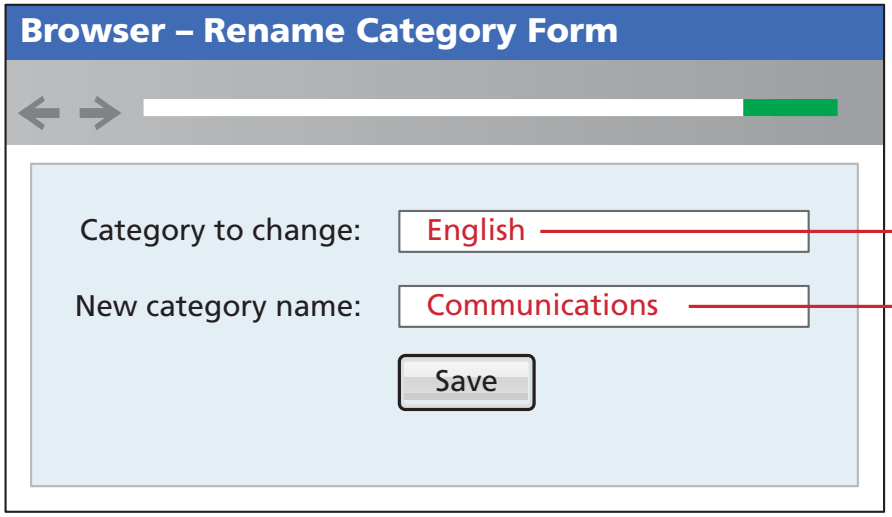
Accessing MySQL in PHP

Using Transactions

```
$pdo = new PDO($connString,$user,$pass);

try {
        // begin a transaction
        $pdo->beginTransaction();
        // a set of queries: if one fails, an exception will be thrown
        $pdo->query("INSERT INTO Categories (CategoryName) VALUES ('Philosophy')");
        $pdo->query("INSERT INTO Categories (CategoryName) VALUES ('Art')");
        // if we arrive here, it means that no exception was thrown
        $pdo->commit();
} catch (Exception $e) {
        // we must rollback the transaction since an error occurred with insert
        $pdo->rollback();
}
```

Accessing MySQL in PHP

Getting user input into a query




```
<form method="post" action="rename.php">
  <input type="text" name="old" /><br/>
  <input type="text" name="new" /><br/>
  <input type="submit" />
</form>
```

$_POST['new']

Communications

$_POST['old']

English

```
UPDATE Categories SET CategoryName='Communications' WHERE CategoryName='English'
```

Accessing MySQL in PHP

Getting user input into a query

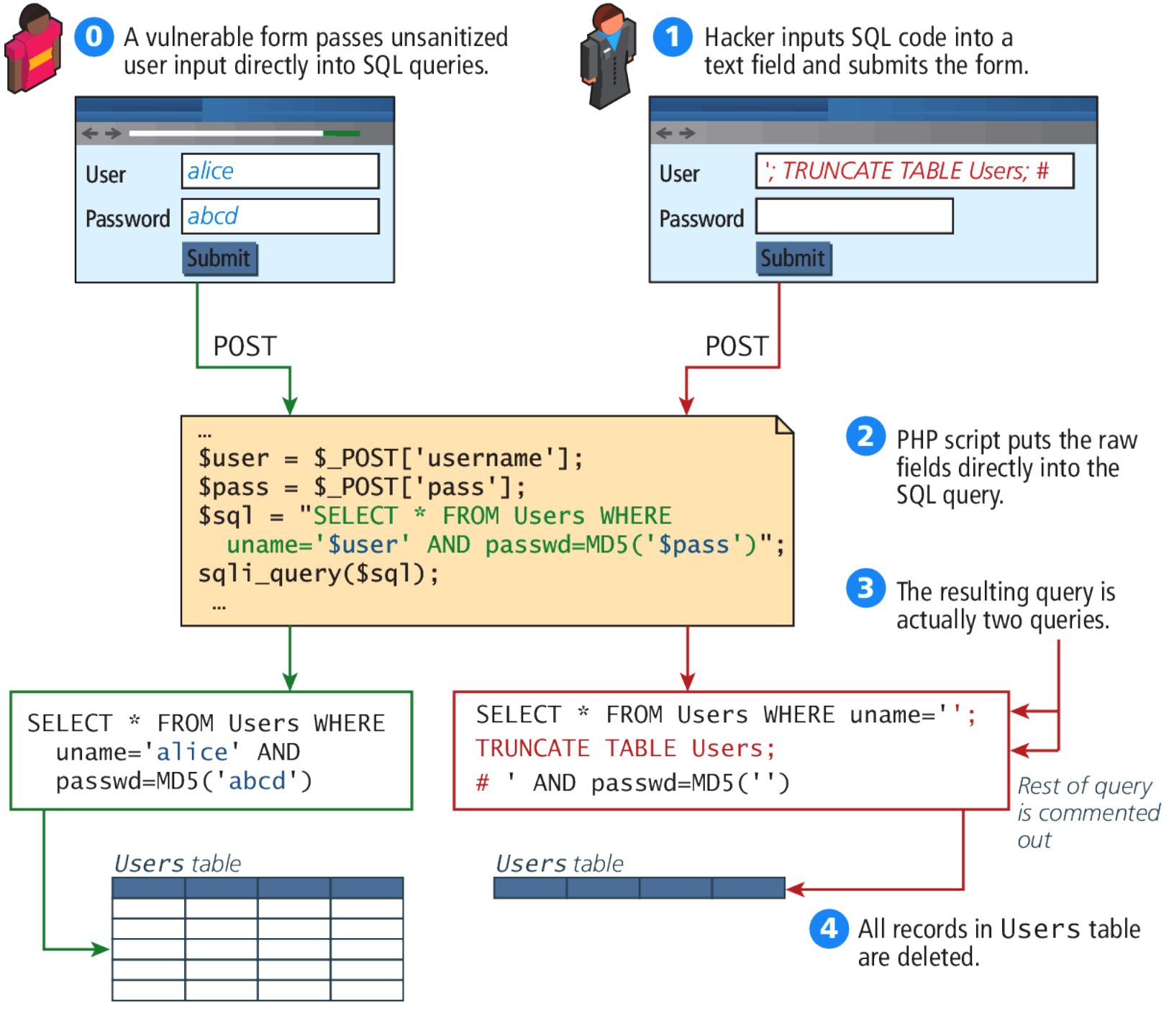
```
$sql = "UPDATE Categories SET CategoryName='Web'
WHERE
        CategoryName='Business'";
$count = $pdo->exec($sql);
echo "<p>Updated " . $count . " rows</p>";
```

Integrating User Data

Not everyone is nice.

```
$from = $_POST['old'];
$to = $_POST['new'];
$sql = "UPDATE Categories SET CategoryName='$to' WHERE
        CategoryName='$from'";

$count = $pdo->exec($sql);
```

While this does work, it opens our site to one of the most common web security attacks, the **SQL injection attack.**


0 A vulnerable form passes unsanitized user input directly into SQL queries.
1 Hacker inputs SQL code into a text field and submits the form.
User
alice
Password
abcd
Submit
User
'; TRUNCATE TABLE Users; #
Password
Submit
POST
POST
...
$user = $_POST['username'];
$pass = $_POST['pass'];
$sql = "SELECT * FROM Users WHERE
uname='$user' AND passwd=MD5('$pass')";
sqli_query($sql);
...
2 PHP script puts the raw fields directly into the SQL query.
3 The resulting query is actually two queries.
SELECT * FROM Users WHERE
uname='alice' AND
passwd=MD5('abcd')
SELECT * FROM Users WHERE uname='';
TRUNCATE TABLE Users;
' AND passwd=MD5('')
Rest of query is commented out
Users table
Users table
4 All records in Users table are deleted.

Defend against attack

Distrust user input

The SQL injection class of attack can be protected against by

- **Sanitizing** user input
- Using **Prepared Statements**

Sanitize User Input

Quick and easy

Each database system has functions to remove any special characters from a desired piece of text. In MySQL, user inputs can be sanitized in PHP using the mysqli_real_escape_string() method or, if using PDO, the quote() method

```
$from = $pdo->quote($from);
$to = $pdo->quote($to);
$sql = "UPDATE Categories SET CategoryName=$to WHERE
        CategoryName=$from";

$count = $pdo->exec($sql);
```

Prepared Statements

Better in general

A **prepared statement** is actually a way to improve performance for queries that need to be executed multiple times. When MySQL creates a prepared statement, it does something akin to a compiler in that it optimizes it so that it has superior performance for multiple requests. It also integrates sanitization into each user input automatically, thereby protecting us from SQL injection.

Prepared Statements

mysqli

```php
// retrieve parameter value from query string
$id = $_GET['id'];

// construct parameterized query - notice the ? parameter
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID=?";

// create a prepared statement

if ($statement = mysqli_prepare($connection, $sql)) {
    // Bind parameters s - string, b - blob, i - int, etc
    mysqli_stmt_bindm($statement, 'i', $id);

    // execute query
    mysqli_stmt_execute($statement);

   // learn in next section how to access the returned data
   …
}
```

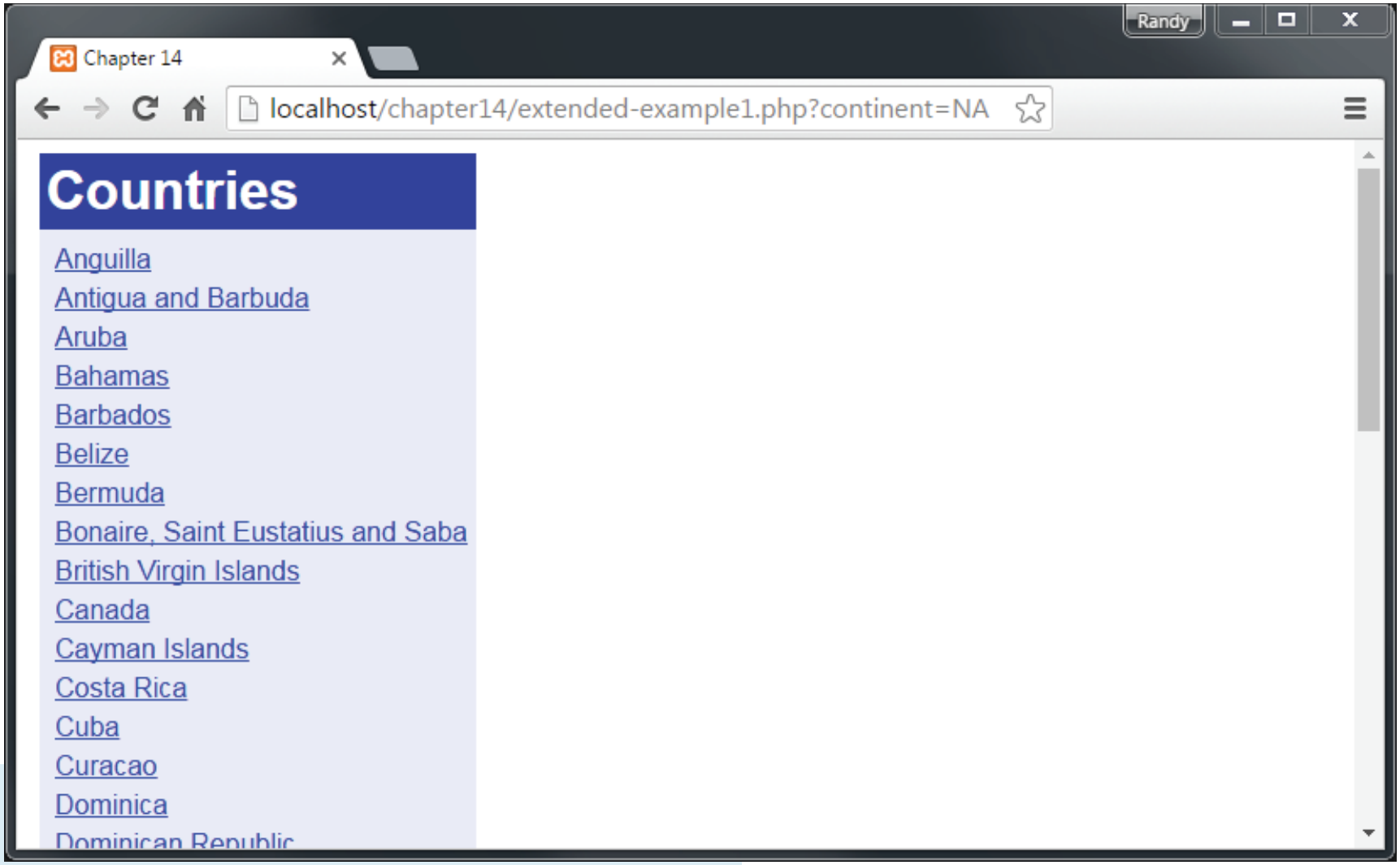
Prepared Statements

PDO

```php
// retrieve parameter value from query string
$id = $_GET['id'];

/* method 1 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $id);
$statement->execute();

/* method 2 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = :id";
$statement = $pdo->prepare($sql);
$statement->bindValue(':id', $id);
$statement->execute();
```

Accessing MySQL in PHP

Advanced example



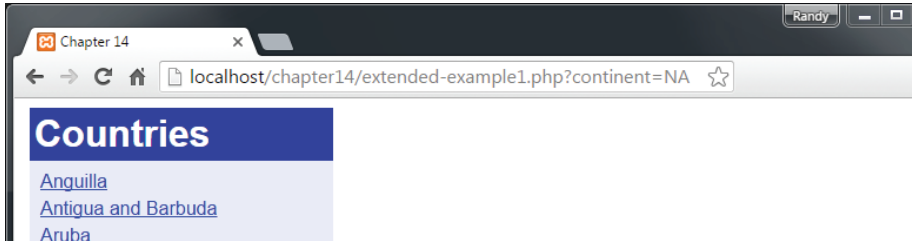
Accessing MySQL in PHP

Advanced example

```php
<?php
// get database connection details
require_once('config-travel.php');

// retrieve continent from querystring
$continent = 'EU';
if (isset($_GET['continent'])) {
   $continent = $_GET['continent'];
}
?>
...
<h1>Countries</h1>
<?php
try {
    $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // construct parameterized query – notice the ? parameter
    $sql = "SELECT * FROM geocountries WHERE Continent=? ORDER BY CountryName ";
    // run the prepared statement
    $statement = $pdo->prepare($sql);
    $statement->bindValue(1, $continent);
    $statement->execute();
    // output the list
    echo makeCountryList($statement);
}
```

config-travel.php

```php
<?php
define('DBHOST', 'localhost');
define('DBNAME', 'travel');
define('DBUSER', 'testuser2');
define('DBPASS', 'mypassword');
define('DBCONNSTRING',
       'mysql:host=localhost;dbname=travel'
?>
```



Accessing MySQL in PHP

Advanced example

```
catch (PDOException $e) {
   die( $e->getMessage() );
}
finally {
   $pdo = null;
}
```


```
function makeCountryList($statement) {
    $htmlList= '<ul>';
    $foundOne = false;
    while ($row = $statement->fetch()) {
        $foundOne = true;
        $htmlList .= '<li>';
        $htmlList .= '<a href="country.php?iso=' . $row['ISO'] . '">';
        $htmlList .= $row['CountryName'];
        $htmlList .= '</a>';
        $htmlList .= '</li>';
    }
    $htmlList.='</ul>';

    if ($foundOne) return $htmlList;
    return 'No countries found';
}
?>
```

Advanced Database in PHP

Summary

Summary

- Databases and Web Development
- Database APIs
- Transactions
- Accessing MySQL in PHP